Appl. Math. and Comp. Sci., 1993, vol.3, No.3, 595–612

FRAMES IN DIAGNOSTIC REASONING

WOJCIECH CHOLEWA*

A frame is a flexible representation of statements and rules. It is especially attractive for the designing of interfaces with an expert system. It has been shown in the paper, that frames are a useful tool in the designing of reasoning systems for technical diagnostics. They can be successfully applied to support the most difficult part of such a task – ie. to acquire knowledge. The main features of an expert system shell composed of a frame interpreter and tools for programmers have been presented, too.

1. Introduction

The detection of changes in the internal state of a machine by means of nondestructive tests is one of the main goals in technical diagnostics. Such a detection is based on the features of interaction between the machine and environment. The basic diagnostic model of a machine can be built using a *set of features*. The methods dealing with the diagnostics of machines are frequently strongly based on experience. Valuable relationships are established intuitively without knowledgeable reasoning or study and corresponding phenomenologic models are used for explanation only.

Technical diagnostics may be supported by expert systems. The basic elements of an expert system are shown in Figure 1. To be effective, expert systems must contain a large amount of domain expertises organized appropriately. The representation of knowledge in expert systems for technical diagnostics differs from representation in medical applications. In technical diagnostics there is a variety of objects, but their structure and actions are more clear and easier to describe than the human body. Most of all, expert systems make use of rules. The designing of large *knowledge–bases containing rules* is a hard work. A lot of difficulties are connected with the direct acquisition of rules from experts (Cholewa and Moczulski, 1990). For a large set of independent rules we can lose track of them. It is strongly inconvenient to check their completeness and consistency. Moreover, although a lot of guidelines exist instructing how to write the documentation for the programs in FORTRAN, PASCAL, C and other languages, it is very difficult to write an effective documentation for the knowledge bases.

* Chair of Fundamentals of Machine Design, Silesian Technical University, 18A Konarskiego Str., 44–100 Gliwice, Poland

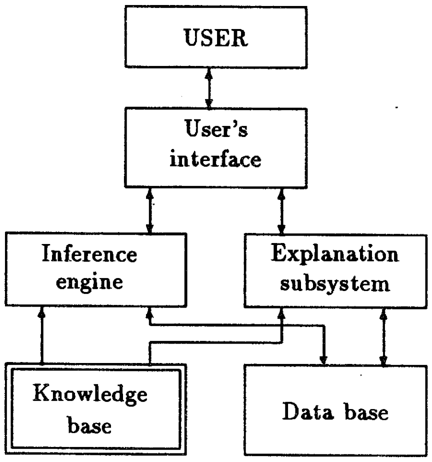

Fig. 1. Basic elements of an expert system.

2. Knowledge Representation in Expert Systems

The process of ***knowledge acquisition*** may be supported by special forms of logs made by experts. Such logs (records) are then processed by a knowledge engineer (or by a special program) to obtain an appropriate set of rules. Among others, such forms as fault trees, test trees, diagnostic graphs and decision tables ought to be recommended. It is necessary to stress that it does not mean that an expert system has to be designed in this way, e.g. as a decision table. From experience it follows only that such approaches are very useful tools for knowledge acquisition.

2.1. Fault Trees

If one element or unit of a machine does not operate correctly, it causes further abnormal actions of some other elements. Finally, failure propagation may result in a general failure of the machine. The simplest way to find out the source of failure is to trace back the *cause–effect* pathway, which may be effectively achieved by means of fault trees.

The fault tree is a well known and widely used tool in the area of reliability analysis and troubleshooting (Brown *et al.*, 1975; Lapp and Powers, 1977; Pau, 1981). To define a fault tree we should identify

- a list of the main elements and/or functional units (subsystems) constituting the machine,

and for each element or unit as well as for the whole machine:

- the expected actions which ought to be performed,
- the failures causing that the machine or unit or element does not perform its actions,

- the conditions causing the failures,
- the elements and/or units contributing to the failure.

Failures and their conditions are interpreted as events. They may be represented in a graph model as nodes, which are connected by *cause–effect* links. The links can be clustered by means of AND, OR, XOR gates. We should distinguish (Innis and Hammond, 1977) the following kinds of nodes (see Fig. 2)

- a primal (basic) event which shows the failure of an element or unit where no further development is needed,
- an undeveloped event which could be developed further on if more information was available and might be used to show the failure of the given subsystem,
- an intermediate event, which is a condition or event caused by other events.

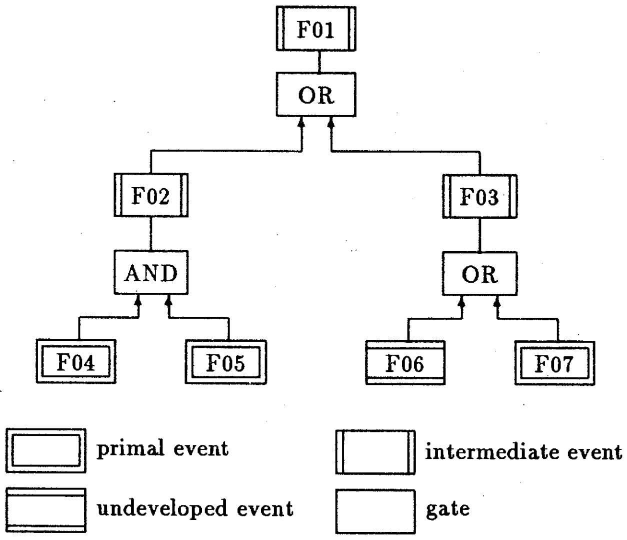

Fig. 2. Shape of a fault tree (an example).

From the fault tree we may infer which primal events or combination of primal events cause the root event. To construct the fault tree we should have a thorough knowledge and experience about the design of the machine and its environment.

The fault tree can be extended to a weighted fault tree. The weightings are assigned to the nodes and/or links. They are understood as probabilities or conditional probabilities of occurrence and may be used for reliability calculations. The main difficulty in the designing of weighted fault trees is caused by the lack of sources of probability values.

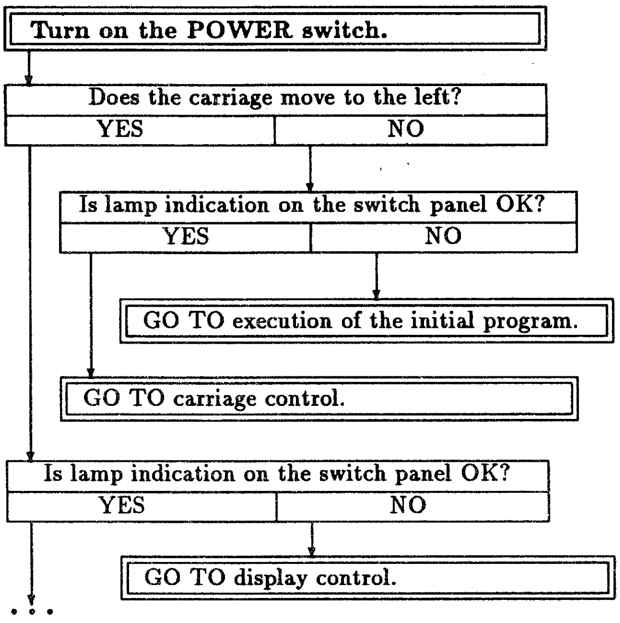
Turn on the POWER switch.
Does the carriage move to the left?
YES
NO
Is lamp indication on the switch panel OK?
YES
NO
GO TO execution of the initial program.
GO TO carriage control.
Is lamp indication on the switch panel OK?
YES
NO
GO TO display control.
...

Fig. 3. Example of a test tree (selected part of the Technical Manual for an Epson printer).

2.2. Test Trees

Fault trees are tools for failure localization. To locate a basic failure we ought to follow a given set of *cause–effect* links. Test trees result from fault trees, but they don't represent directly the causal links between the failures (see Fig. 3). Their nodes contain descriptions of procedures or tests which have to be made. They may be written as weighted graphs. Weights give the measures of cost or time necessary for the diagnosis. The sequence of diagnostic or check tasks may be optimized. For a review of related techniques see (Pau, 1981).

2.3. Diagnostic Graphs

The basic problems concerning the designing of diagnostic tests are connected with methods of concluding about the relations between signal features and the state of the object.

A fault tree represents causal dependencies between the faults. A test tree can be designed for a selected group of machines. Both kinds of trees don't include explicit information about the relations between the state of the object and the features of diagnostic signals.

A *diagnostic graph* contains nodes representing the classes of values of the signals. These classes are related to the faults on the fault tree (see Fig. 4) basing on our experience. The links of nodes in the fault tree are then plotted in a diagnostic graph. In general, a diagnostic graph is not a tree. Selected nodes from this graph form a set of features that are continuously monitored.

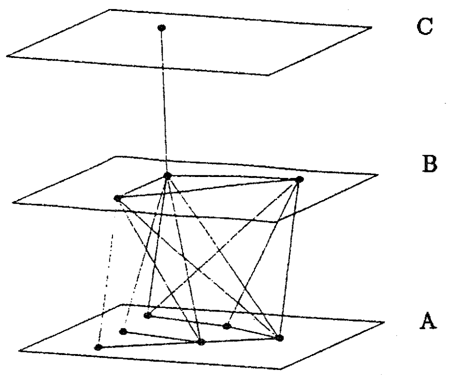

Fig. 4. The idea of the diagnostic graphs. A: fault tree, B: diagnostic graph, C: monitored features.

The diagnostic process may be carried out in the following two steps:

- unexpected values of monitored signals are detected,
- from the diagnostic graph we can obtain the necessary paths between different features of signals which ought to be taken into account in discovering the faults.

2.4. Decision Tables

A *decision table* is a very useful tool for the designing of reasoning systems for technical diagnostics. A basic worksheet for the decision table consists of three main parts (see Fig. 5)

- rows $C1$, $C2$, ...: conditions, questions, tests,
- rows $A1$, $A2$, ...: actions and/or conclusions,
- rows $E1$, $E2$, ...: exits.

Each column on the right–hand side of the table contains the definition of a rule. Entries of conditions for these rules consist of answers to questions or test results. It has to be assumed (Hurley, 1983), that:

- The execution order of actions is given by numbers, where actions marked by the same numbers (or by X) are executed in top–to–bottom order.
- Rules are tested from left to right, until all the conditions are fulfiled. It means that not all rules will be tested.
- For a decision table to be complete each possible combination of entries for the given conditions has to be included once and only once.
- Missing rules, redundant rules and conflicting rules are not accepted.

For a given table it is possible to have several different exits but only one exit in each rule. There are several sorting and optimizing criteria for decision tables which can be defined.

Sleeve bearing		Rules						
		1	2	3	4	5	6	7
$C1$	Self–excited vortex vibration with a frequency of about one–half the speed of the rotating shaft are encountered.	Y	N	–	N	N	Y	Y
$C2$	Temperature of the oil is too high.	Y	Y	Y	N	–	–	N
$C3$	Temperature of the oil is too low.	–	–	Y	N	Y	Y	N
$C4$	Bearing has a lemon–shaped working surface of the sleeve.	–	–	–	–	–	–	N
$A1$	Radial clearance of the sleeve bearing ought to be reduced.						X	
$A2$	Radial clearance of the sleeve bearing ought to be increased.		X					
$A3$	The amount of oil that must be pumped through the bearing ought to be decreased.					X		
$A4$	Thermal calculations of the bearing ought to be performed.	X						
$A5$	Lemon–shaped working surface of the sleeve ought to be introduced.							X
$E1$	Error, RUN_AGAIN			X				
$E2$	QUIT	X	X		X	X	X	X

Fig. 5. Limited entry decision table.

2.5. Frames

A *frame* is a description of a real or an abstract object and can be used as a flexible representation of conditions and actions in decision tables. The notion of frames was introduced by Minsky (1975) (see also Bartlett, 1932; Goffman, 1974). His basic goal was concerned with the designing of a data–base containing encyclopedic knowledge, needed in commonsense reasoning.

A frame contains *slots* representing attributes of the object (Fig. 6). Slots contain *facets* connected with values, default values and/or procedures (called demons) by which the values may be obtained. It is important to point out that each facet can contain values or demons. Such an inclusion of demons in frames joins procedural and declarative representations. Some systems distinguish between frames for classes and frames for individuals.

Frames may be arranged in ***hierarchical structures*** (see Fig. 7) which make it possible to develop and process the idea about classes without being disturbed by details of any particular object. Such structures are expressed by links, called AKO (a kind of), between superframes (parent frames) and subframes (derived frames).

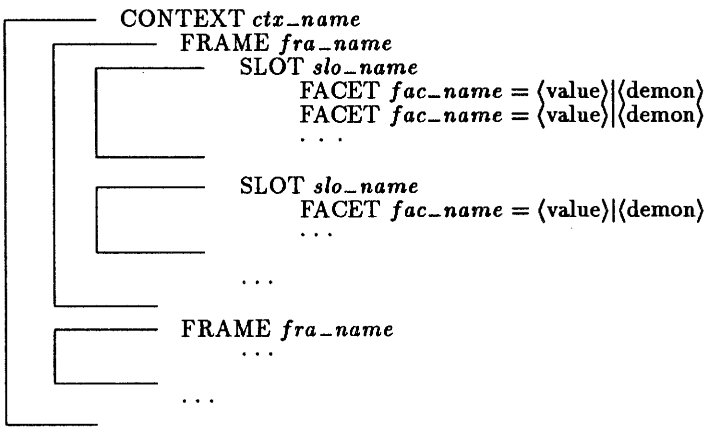

Fig. 6. Elements of a frame.

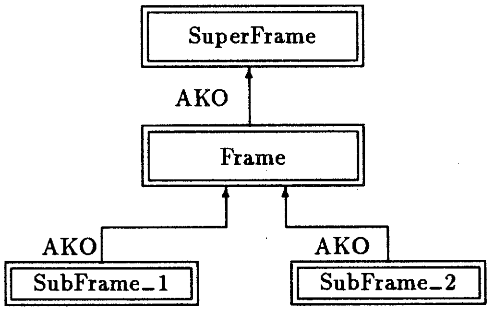

Fig. 7. Simple hierrarchies (there exists only one superframe for each frame).

Searching for a slot value for a frame is the basic task in frame systems. Special properties are assigned to the facets: *value, if_needed, if_added* and *if_removed.* The slot value is assigned to the facet *value.* When we are looking for the value of the slot, the content of the facet *value* is returned. If such a facet is missing, the facet *if_needed* points to the value or to a demon returning the value. Slots that are not present in a frame are inherited from superframes. Searching returns alternatively:

- a list containing values of the same slots in all superframes,
- only the first value that is found; this value overrides the values in other superframes.

Inheritance is the most important feature of frames, which makes it possible to eliminate a redundancy of data and to handle exceptions. It can also be used to generate reasonable default data or assumptions in case of incomplete information. Special facets, such as *if_added* and *if_needed* may be applied for forward and backward chaining, respectively. Simple hierarchy results in a tree structure of frames (Fig. 7). For such structures a search path is given by the AKO links, starting from the selected node upwards and the search time grows, at worst, linearly with the number of nodes. More advanced multiple inheritance (Fig. 8) results in a directed acyclic graph, for which the strategies for ***depth first*** or ***breadth first*** searching ought to be applied, where:

- the results depend strongly on the arrangement of superframes,
- the search time grows, at worst, exponentially with the number of nodes,
- redundant searches may be expected (for depth–first search).

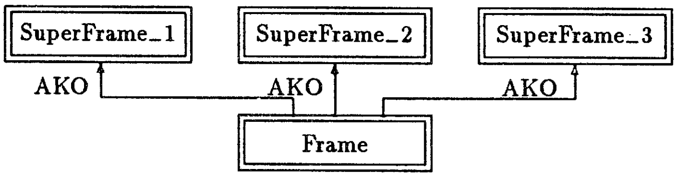

Fig. 8. Multiple inheritance (each frame can possess a few superframes).

Encapsulation is a property postulated by object–oriented–programming. It states that the data structures and procedures dedicated to manipulate the data ought to coupled together and ought to be isolated in some degree from the direct access by other procedures. Frames allow us to control the degree of encapsulation.

Frames are an example of object–oriented programming, which makes it possible to generalize, classify and generate abstractions. Frames offer high computational efficiency. They are an interesting tool for the designing of interfaces with users and with external sources of data (e.g. measuring devices).

3. Statements and Rules

Most expert systems use a ***knowledge base***. It may be assumed that a knowledge base is a collection of statements describing relationships between entities of the real world as well as abstract concepts. Such statements are variously called sentences, clauses, formulas and most often facts. Of course, from the direct use of the notion *fact* there may result a lot of misinterpretations, because the particular statements (describing the real world) are not independent, real existing facts. They are only some belief that something has happened or has been done.

A common way to represent statements is an *object–attribute–value* triple, used e.g. in the well known pattern expert system MYCIN (Shortliffe, 1976). *Attributes* are general characteristics of properties possessed by *objects*. The *value* specifies the particular nature of a property in a given situation. Of course, the sets of such triples are flat (they contain no underlying structure) and the maintenance of great sets is extremely difficult. A dozen of different ideas of structuring the knowledge base has been proposed and implemented (see e.g. Jackson, 1986). Very important is the idea of frames.

3.1. Rules

Mathematical logic was one of the first formalism that was proposed as a knowledge representation. The inference in logic is connected with the notion of implication. If p and q are given statements, then the implication $p \longrightarrow q$ is true when q is true or p is false. Implication $p \longrightarrow q$ is often written as the following rule:

$$\text{if } p \text{ then } q \tag{1}$$

To allow an appropriate explanation facility the knowledge base ought to contain the rules in an extended form

$$\text{if } p \text{ then } q \text{ because } \textit{explanation} \tag{2}$$

The classical logic uses two basic patterns for the deduction in propositional calculus:

- modus ponens (rule of detachment) – for statements p and q, if the rule $p \longrightarrow q$ is true and p is true, then q is true; we can write this in the following form

$$\begin{array}{ll} p \longrightarrow q & \text{rule} \\ p & \text{premise} \\ \hline q & \text{conclusion} \end{array} \tag{3}$$

- modus tollens (rule of contrapositive) – for statements p and q, if the rule $p \longrightarrow q$ is true and q is false, then p is false; we can write this in the following form

$$\begin{array}{ll} p \longrightarrow q & \text{rule} \\ \neg q & \text{premise} \\ \hline \neg q & \text{conclusion} \end{array} \tag{4}$$

Example.

(1)
$$\begin{array}{l} \textbf{if } \textit{it is raining } \textbf{then } \textit{roads are wet} \\ \textit{it is raining} \\ \hline \textit{roads are wet} \end{array}$$

(2)
$$\begin{array}{l} \textbf{if } \textit{it is raining } \textbf{then } \textit{roads are wet} \\ \textit{roads are not wet} \\ \hline \textit{it is not raining} \end{array}$$

■

If we know only that p is false (or respectively that q is true) we are not able to arrange a reliable inference about the logical value of q (or respectively p). A lot of misunderstandings are connected with the interpretation of the implication. It is necessary to point out that the implication $p \longrightarrow q$ doesn't state that q follows from p; e.g. the both implications

$$(2 > 1) \longrightarrow (3 > 0) \quad \text{and} \quad (2 < 1) \longrightarrow (3 > 1)$$

are true. Moreover, it is important that we are able to draw reasonable conclusions only when the implication is true because from the assumption that an implication $p \rightarrow q$ is false it follows only that p has to be true and q has to be false. From the last remark it follows that the knowledge base ought not to contain the rules in a form

It is not true that **if** p **then** q.

The conditional part (left–hand side) of a rule may include the composed statements. In pure logic it is enough to consider only two kinds of such composition, namely conjunction ($p = p_1$ AND p_2) and disjunction ($p = p_1$ OR p_2). In each real application we can easily obtain the situation where the logical value of a statement results independently from few statements (e.g. opinions from different experts). Although it seems to be a case of redundancy in expert systems, we are not allowed to reduce this kind of overloading because designing the knowledge base we don't know which sources of information will be available for the user. In such circumstances the conjunction and disjunction are irrelevant operators and we have to introduce the next one called aggregate ($p = p_1$ AGG p_2). The properties of such operator are discussed e.g. by Cholewa (1985).

The rules contained in the knowledge base can be driven (activated) forward from those statements that we know to be true towards unknown statements or they can be driven backward from the statements (hypothesis) that we wish to establish the statements necessary for their truth. It is important to make a clear distinction between forward/backward chaining of rules and forward/backward solving strategies used by the expert system (forward/backward reasoning). In addition to distinguishing between forward/backward chaining and reasoning, we also need to distinguish between depth–first and breadth–first search of rules in a knowledge base.

It seems today that experts can express most of their problem–solving techniques as a set of condition–action rules. Rules if *premise* then *conclusion* can be very easily extended into production rules if *condition* then *action*. Such rules provides an extremely powerful model for human thought and allow to represent the knowledge about how to carry out the reasoning. For more details see e.g. (Brownston *et al.*, 1985; Hayes–Roth *et al.*, 1983; Jackson, 1986).

3.2. Approximate Rules

Diagnostic knowledge results from the experiences of experts. It is not given in an rigorous form and we often have to deal with rules which are true in most (but not in all) cases. It means that the statements and rules in such applications are often uncertain and/or unprecise. Approximate statements can be represented in a lot of different ways, where certain rules and certain statements can be always taken into account as a special case of approximate one. Several ad hoc, empirical and theoretically based approaches to represent approximate statements and rules are known (see e.g. Dubois and Prade, 1988; Zadeh, 1983; Shortliffe, 1976; Kruse and Meyer, 1987).

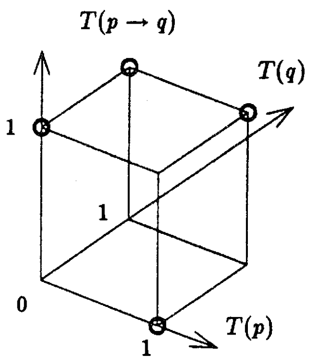

Fig. 9. Two–valued implications.

The simplest approach is the direct application of probability theory and standard Bayesian model. A modification of the probability theory results in the ***truth values*** $T(s)$ from the range [0,1] (or $[-1,1]$), assigned to each of the statements. They are interpreted as an extension of two logical values NO=0 and YES=1, onto the ordered set [0,1] of real numbers. Particular implementations differ mainly in the interpretation of the value $T(s) = 0$, which can point statements that are false or which can point only the statements for which we haven't any source of information that they are true. The last case doesn't mean that there exist some reasons to interpret such statements as false.

For the reasoning by means of truth values we need an extension of implication. The two–valued implication is shown in Figure 9. To extend the definition into a definition of continuous implication which is necessary for dealing with truth values $T(s) \in [0,1]$ we have to define a surface (a function) spanned on the points shown in Figure 9. Some examples are (see Fig. 10):

$$T(p \to q) = \begin{cases} 1 & \text{if} \quad T(p) \leq T(q) \\ T(q) & \text{otherwise} \end{cases} \tag{5}$$

$$T(p \to q) = \max(1 - T(p), T(q)) \tag{6}$$

$$T(p \to q) = \min(1 - T(p) + T(q), 1) \tag{7}$$

On the basis of such implications we can generalize modus ponens and modus tollens (see e.g. Dubois and Prade, 1988).

The first well known expert system (designed for medical applications about 1972) which uses the uncertain statements and uncertain rules is MYCIN. For a given evidence e (reference statement) and a hypothesis h (statement to be evaluated) MYCIN introduces the following three measures:

- measure $\mathrm{MB}(h, e)$ of belief in the hypothesis h,
- measure $\mathrm{MD}(h, e)$ of disbelief in the hypothesis h,

- certainty factor CF(h, e) considered as a truth value of the hypothesis h, and indicating a predominance of confirming (positive value) or predominance of opposing (negative value) evidence.

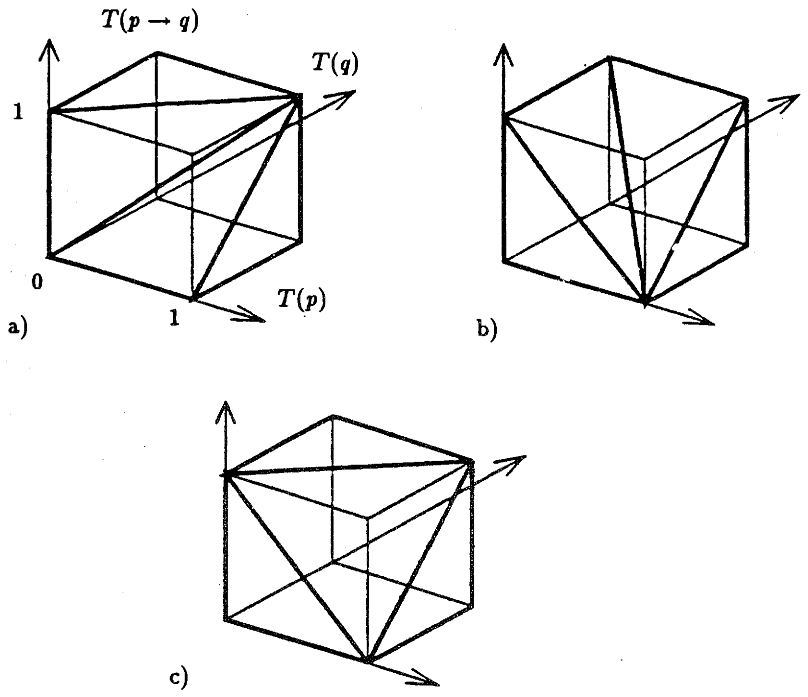

Fig. 10. Examples of implication: a) (5), b) (6), c) (7).

Measures of belief and disbelief are interpreted as some kinds of conditional probability, where

$$MB(\neg h, e) = MD(h, e) \tag{8}$$

$$CF(h, e) = MB(h, e) - MD(h, e) \tag{9}$$

MYCIN uses the following formulas

$$\begin{aligned} MD(h_1 \wedge h_2, e) &= \max(MD(h_1, e), MD(h_2, e)) \\ MB(h_1 \wedge h_2, e) &= \min(MB(h_1, e), MB(h_2, e)) \\ MD(h_1 \vee h_2, e) &= \min(MD(h_1, e), MD(h_2, e)) \\ MB(h_1 \vee h_2, e) &= \max(MB(h_1, e), MB(h_2, e)) \end{aligned} \tag{10}$$

The most interesting is that $MB(h, e)$ and $MD(h, e)$ allow us to take into account separately all premises *pro* and *contra* and we are able to distinguish between the two cases:

- we haven't any sources of information about the logical value of hypothesis,
- known premises *pro* are compensated by known premises *contra*.

We are not able to make such distinctions when we use only the single value (e.g. truth value) assigned to the statement, because the value $T(s) = 0.5$ maps the both cases.

3.3. Possibility and Necessity

An interesting modification of reasoning patterns was obtained by means of modal logic. The notions of possibility and necessity form concept for the measures of possibility $\Pi(s)$ and necessity $N(s)$, assigned to statements. Leaving out the rigorous explanation we can interpret the values of these measures as boundaries of a hypothetical range for the unknown truth value

$$0 \leq N(s) \leq T(s) \leq \Pi(s) \leq 1 \tag{11}$$

By means of $N(s)$ and $\Pi(s)$ we can distinguish the case of compensated premises pro and contra $N(s) = \Pi(s) = 0.5$ from the case with a lack of information $N(s) = 0$ and $\Pi(s) = 1$. Some extensions of modus ponens and modus tollens were proposed (see Dubois and Prade, 1988):

$$\begin{array}{l} N(p \rightarrow q) \geq a \\ N(p) \geq b \\ \hline N(q) \geq \min(a, b) \end{array} \tag{12}$$

$$\begin{array}{l} N(p \rightarrow q) \geq a \\ \Pi(p) \leq b \\ \hline \Pi(p) \leq \max(1 - a, b) \end{array} \tag{13}$$

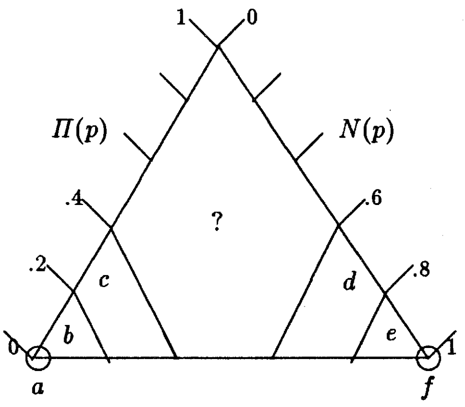

Fig. 11. Plot of an approximate statement:
a) certainly NO, b) perhaps NO, c) maybe NO,
d) maybe YES, e) perhaps YES, f) certainly YES.

The result of reasoning carried out with the use of possibility and necessity may be mapped on the diagram in the form of a triangle (see Fig. 11). Such diagram allows to introduce the linguistic descriptions for the selected pairs of values (N, Π). This simplifies the dialogue with users because the verbal descriptions of certainty are more user–friendly than numbers.

3.4. Bilateral Implication

There exist a lot of examples of dependences between the state of an object and features of diagnostic signals. They can be interpreted as the following ordered relations:

from the known state it *follows* a special property of the signal.

Of course, we are not able to use such relation directly for the robust reasoning in the diagnostics because it is highly possible that the same special property of a signal follows from an another state of the object. Moreover, we have often no reasons for assuming that the discussed relation is a causal relation. To represent correctly all cross–dependences we ought to write the relations in such a form that both modus ponens and modus tollens may be applied together. It can be done by means of bilateral implication (Cholewa, 1986). The bilateral implication $p \leftrightarrows q$ is simply a pair of both underlaying implications $p \rightarrow q$ and $q \rightarrow p$. The implication has to be symmetric for modus ponens and tollens because the result of reasoning ought to be independent from the particular forms of basic statements:

- statement 1: *object X has the property A*,
- statement 2: *object X has the property $\neg A$*,

and questions sent to the user.

The notion of necessity and possibility and the concept of bilateral implication can be used together. It results in the following joint (generalized) modus ponens and tollens

$$\begin{array}{l} T(p \rightarrow q) \\ T(q \rightarrow p) \\ N(p), \Pi(p) \text{ such that } N(p) \leq T(p) \leq \Pi(p) \\ \hline N(q) \leq T(q) \leq \Pi(q) \end{array} \tag{14}$$

where e.g. for the Lukasiewicz's implication (7) we have

$$\begin{aligned} N(q) &= \max(0, N(p) + T(p \rightarrow q) - 1) \leq N(p) \\ \Pi(q) &= \min(1, \Pi(p) - T(q \rightarrow p) + 1) \geq \Pi(p) \end{aligned} \tag{15}$$

The reasoning process can be mapped on the diagram (Fig. 11), as a sequence of vectors going up. It is easy to see that the certainty of the conclusion can not be better then the certainty of the premise. To improve this situation, that means to obtain conclusions that are more certain than the premises, we ought to have some set of independent rules resulting in the same conclusion obtained as an aggregate of partial conclusions.

3.5. Compound Decision Tables

Decision tables can be used directly to model systems with forward chaining. After slight modifications of exits, i.e. assuming that each table ought to return a value (e.g. YES or NO) we can speak about such tables as about special cases of conditions. Return values have to be assigned to all columns on the right-hand side of the table (which are terminated with RETURN). Moreover, a default value ought to be assigned to the whole table in case of incomplete sets of rules. The table returning a value can be used as a condition in another table. It is very useful to write such a table as a special case of a frame. The table entries ought to be written as frames, too (see Fig. 12).

table$_k$ (object$_k$, attribute$_k$, value$_k$)		Rules		
		$R1$	$R2$	
C_1				
...				
C_i	statement$_i$ (object$_i$, attribute$_i$, value$_i$)			
...				
C_j	table$_j$ (object$_j$, attribute$_j$, value$_j$)			
...				
A_1				
...				
	RETURN max$(C_1, ..., C_i, ..., C_j, ...)$ *	v_1	v_2	

Fig. 12. Compound decision table.

Such modifications allow us to:

- simulate some kinds of backward chaining,
- write the rules dealing with the knowledge (about the diagnosed objects) and with the meta-knowledge (about the reasoning process) in a similar, uniform way.

3.6. Rule-Based or Case-Based

Decision tables have been proposed as a special tool for writing rules. It is known, that a direct acquisition of rules from a human expert is quite difficult and connected with the possibility of erroneous constraints. More convenient (and more easy) for the designer of an expert system, as well as for the final user, is the acquisition of rules not by direct specification but indirectly (in a hidden form) by means of examples. Such examples result from case studies and point out the pairs connecting the evaluations of technical states with symptoms. Examples may be written in a decision table in a similar way as rules.

To apply such an extension of decision tables it is necessary to change some assumptions about the tables, namely:

- rules ought to be tested from left to right and all rules have to be tested,
- for a decision table a combination of entries for the given conditions can be repeated many times,
- missing rules, redundant rules and conflicting rules are allowed.

4. System MAS

Frames and decision tables may be realized by means of many tools, where object oriented programming is most important. Maintenance Aid Shell MAS (Cholewa, 1992) is an expert system shell (an extension of VV_SHELL (Cholewa *et al.*, 1991)) containing the production system, frame interpreter, frame editor, browser/debugger, reference data base and an interface for extracting the diagnostically useful statements from the data in the data base. The interface isolates the actual knowledge base of MAS from any paticular machinery data base and can easily be changed to process the data derived from virtually any condition monitoring system provided by the data base. The reference data base contains the information required by the reasoning system and concerning the configuration of the machine train.

MAS runs on the IBM PC family of computers under MS Windows. The frame interpreter of MAS is a processing unit specially designated to handle the different types of statement structures, represented by means of frames. It can also handle so called approximate statements and cooperate with other reasoning systems (Cholewa and Czogała, 1991; Czogała and Cholewa, 1991). LISP–like frame description language allows to take into account different kinds of inheritance of frames. E.g. hierarchical links between frames can be listed as follows:

```
FRAME SubFrame_1                                    /* see Figure 7 */
    SLOT ako
        value = (Frame)
FRAME Frame                                         /* see Figure 8 */
    SLOT ako
        value = (SuperFrame_1 SuperFrame_2 SuperFrame_3)
```

Examples of special functions included in the frame interpreter of MAS:

- frame–processing functions
  (DEL *fac*), (DEL_CON *ctx*), (DEL_FRA *fra*), (DEL *slo*), (FACET ...), (FACET_V ...), (GET *slo*), (GET_INH *slo*), (SET *fac val*), (SLOT ...), (SLOT_V ...), (VIEW *slo*), (VIEW_INH *slo*), . . .
- task–arranging functions
  (BREAK), (EVAL *lstA lstB lstC*), (IF *cond val1 val2*), (RUN *ctx*), (WHILE *cond val*), . . .

- list-processing functions (HEAD *lst*), (SEL *lst pos*), (TAIL *lst*), . . .
- uncertainty processing functions (F_AGR *fuz1 fuz2*), (F_AND *fuz1 fuz2*), (F_IMP *fuz1 fuz2*), (F_NOT *fuz1*), (F_OR *fuz1 fuz2*), . . .
- user's interface functions (CONFIRM *txt*), (DISPLAY *val1 val2* ...), (MENU *topic default* (*str1 val1*) (*str2 val2*) ...), (PROMPT *default ask topic*), . . .

All the elements of frames in MAS are identified by their names (see Fig. 6). The names ought to be locally unique. It means, e.g., that all slots in a given frame ought to possess individual, different names. Global uniqueness of names is not required, i.e. we can use the same name for slots in different frames. Such an assumption results in a polymorphism – the names are shared and their meaning depends on the given context.

The frame interpreter of MAS enables us to control the degree of encapsulation. This is achieved by the using of demons (making no difference between data and a description of data ie. code) and by one of the following two possibilities to point out a slot:

- slot may be pointed out by its name,
- slot may be pointed out by its qualified name, i.e. by the pair composed of the name of the slot and the name of a frame to which this slot belongs.

In the first case we obtain an access to the slot (for which we are looking) in the current frame. In the second case an access is obtained to a particular slot in a given frame. This allows us to use some slots as global and some slots as local (private) ones. In both cases all assumptions about the inheritance are valued.

References

Bartlett F.C. (1932): *Remembering.* — Cambridge: University Press.

Brown J.S., Burton R.R. and Bell A.G. (1975): *SOPHE – A step toward creating a reactive learning environment.* — Int. J. Man–Mach. Stu. No.7, pp.675–696.

Brownston L., Farrell R., Kant E. and Martin N. (1985): *Programming Expert Systems in OPS5.* — Addison–Wesley, Reading MA.

Cholewa W. (1985): *Aggregation of fuzzy opinions – an axiomatic approach.* — Fuzzy Sets and Systems, No.17, pp.249–258.

Cholewa W. (1986): *Reciprocal fuzzy implication.* — First Joint IFSA–EC and EURO–WG Workshop *Progress in Fuzzy Sets*, Warszawa, Poland, Abstracts, pp.19.

Cholewa W. *et al.* (1991): *VV_SHELL User's Guide and Reference Manual.* — Gliwice: Silesian Technical University Press, (in Polish).

Cholewa W. (1992): *Maintenance Aid Shell.* — Report RMT6076, Gliwice: Silesian Technical University Press, (in Polish).

Cholewa W. and Czogała E. (1991): *Management of statements in frame interpreter of CC_SHELL.* — BUSEFAL 49, pp.40–49.

Cholewa W. and Moczulski W. (1990): *Expert systems in technical diagnostics.* — The Essence of Operation, No.2–3 (82–83), pp.331–342; and *Expert Systems in Technical Diagnostics.* — Design Principle, No.4, (84), pp.519–527, (in Polish).

Czogała E. and Cholewa W. (1991): *Uncertainty treatment in a fuzzy production system of CC_SHELL.* — BUSEFAL No.48, pp.124–131.

Dubois D. and Prade H. (1988): *Possibility Theory – An Approach to Computerized Processing of Uncertainty.* — New York: Plenum Press.

Goffman E. (1974): *Frame Analysis.* — New York: Harper & Row.

Hayes–Roth F., Waterman A. and Lenart D.B. (1983): *Building Expert Systems.* — Addison–Wesley, Reading MA.

Hurley R.B. (1983): *Decision Tables in Software Engineering.* — New York: Van Nostrand Reinhold Comp.

Innis Ch.L. and Hammond T. (1977): *Predicting mechanical design reliability using weighted fault trees.* — In: Failure Prevention and Reliability. New York: ASME Press, pp.213–228.

Jackson PP. (1986): *Introduction to Expert Systems.* — Wokingham: Addison–Wesley.

Kruse R. and Meyer K.D. (1987): *Statistics with Vague Data.* — Braunschweig: Reidel Publ. Comp.

Lapp S.A. and Powers G.J. (1987): *Computer-aided synthesis of fault–trees.* — IEEE Trans. Reliability, No.26, pp.2–12.

Minsky M. (1975): *A framework for representing knowledge.* — In: Computers and Thought. (Ed.) Winston P.H., New York: McGraw–Hill, pp.211–277.

Pau L.F. (1981): *Failure Diagnosis and Performance Monitoring.* — New York: Marcel Dekker, (1975, in France).

Shortliffe E.H. (1976): *Computer–Based Medical Consultation MYCIN.* — New York: Elsevier.

Zadeh L.A. (1983): *The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems.* — Elsevier Science Publishers (North–Holland).

Received February 24, 1993
Revised June 21, 1993